Algorithms for the Calzedonia workload allocation problem

Maria Battarra[a], Federico Fraboni[b], Oliver Thomasson[a], Güneş Erdoğan[a], Gilbert Laporte[c], Marco Formentini[d]

[a]School of Management, University of Bath, Bath, BA2 7AY, United Kingdom;
[b]Calzedonia, Via Monte Baldo, 20, Villafranca di Verona, Italy;
[c]HEC Montréal, 3000, chemin de la Côte-Sainte-Catherine Montréal, H3T 2A7 Canada;
[d]Audencia Business School, 8, Route de la Jonelière, 44312 Nantes, France

ARTICLE HISTORY
Compiled April 5, 2020

ABSTRACT
The Workload Assignment Problem consists of assigning a sequence of $|S|$ operations to workers. The order of these operations is fixed. Each operation consists of a batch of B units, hence a total of $|J|$ jobs have to be performed. Each worker is assigned to an ordered subset of consecutive jobs. Workers have different skills, and therefore jobs take a variable time to process, depending on the assigned worker. The study of this problem is rooted in the operations of Calzedonia. In this paper, we briefly introduce the application before presenting algorithms for solving the problem exactly and heuristically. Our computational results compare the performance of a stand-alone mathematical formulation solved by CPLEX, a sequential exact algorithm, and a metaheuristic, with a simple heuristic implemented in the company.

KEYWORDS
Combinatorial optimisation; production; scheduling

1. Introduction

We introduce the Workload Assignment Problem (WAP) which consists of assigning an ordered sequence of $|S|$ operations to a set of workers. The order of the operations is fixed, and each operation consists of a batch of B units, hence a total of $|J| = B \times |S|$ jobs have to be performed. A worker is assigned to exactly one set of consecutive jobs. Workers have different skills, and therefore the time to complete jobs differs among workers. However, contiguous jobs may take the same time if performed by the same worker and if they belong to the same *operation*. Operations are the production steps necessary to complete a product. Each operation has to be performed once on each product of the batch, before sequentially moving to the next operation. A worker is assumed to take the same time to complete the

CONTACT Maria Battarra. Email: m.battarra@bath.ac.uk

same operation on each product in the batch. A *job* is defined as the process of completing one of the operations on one product in the batch. The objective is the minimisation of the longest *workload* among the workers, where the workload is defined as the sum of the processing times of all jobs assigned to a worker.

The closest problem in the literature is the Assembly Line Worker Assignment and Balancing Problem (ALWABP), introduced by Miralles et al. (2008). Our problem differs from ALWABP since it involves batch production and a precedence graph that reduces to a line. These differences allow us to develop exact solution methods with higher efficiency than the available ones.

This work is motivated by our collaboration with Calzedonia, an Italian apparel company. The WAP reflects the workload allocation practice in some of their production sites. We present two exact algorithms and a metaheuristic, and compare our results with those of a heuristic implemented at the company. We show the effectiveness of our metahueristic, with its short CPU time requirement and superior performance.

A formal definition of the WAP is given in Section 1.1. A description of the factory settings that motivated this problem is provided in Section 1.2. Similar scheduling problems are reviewed in Section 2. Section 3 presents a valid mathematical formulation for the WAP. Section 4 describes our sequential exact algorithm. Section 5 describes the heuristic algorithm that was implemented by Calzedonia, as well as our metaheuristic. Computational results are given in Section 6, followed by conclusions and future research directions in Section 7.

1.1. Problem definition

A set of workers W is responsible for the sewing operations of a single type of product on a production line. Each product is manufactured by completing an ordered sequence S of sewing operations. The workers manufacture the product in batches of size B. In order to complete a batch of products, an ordered set of jobs J has to be performed, where $|J| = B \times |S|$ (i.e., all operations in order, on all products in the batch). Each worker k requires t_{ik} units of time to complete job i. However, $t_{ik} = t_{jk}$ if jobs i and j belong to the same operation. The problem is to assign each worker to one contiguous subset of J, so that every job is assigned to exactly one worker. Note that jobs cannot be shared between workers, whereas the jobs belonging to the same operation can be shared among neighbouring workers.

The objective is to minimise the time for the slowest worker to complete their assigned jobs ($\max_{k \in W} \sum_{i \in J} t_{ik} x_{ik}$, where $x_{ik} = 1$ if worker k performs job i, 0 otherwise), which reduces the likelihood of bottlenecks, promoting a smooth flow of products in the line (or a well-paced line). This objective, which we define as z, guarantees that a batch of products is completed every z units of time. It is typically referred to as 'minimising the maximum process time' (Li et al., 2015, 2017). We therefore use this descriptor for our objective in the remainder of this paper. Finally note that the production is repetitive (i.e., a new batch

of products starts being processed whenever the first worker completes his or her jobs from the precious batch), however our analysis can focus on the workload allocation for a single batch without any loss of generality.

1.2. Motivation

Calzedonia is an Italian fashion company established in 1986 that specialises in hosiery, garments, and beachwear for women, men, and children. The company is structured vertically, i.e., the design, production, and distribution of products are handled either directly, or through affiliates. Calzedonia sells its products in over 30 countries, and employs approximately 26,000 workers. Their factories are located in Bulgaria, Croatia, Romania, Ethiopia, Serbia, and Sri Lanka. This research originated in one of Calzedonia's Sri Lankan factories which specialises in bra production.

Figure 1.: Bra production in the Calzedonia Sri Lankan factory

The Sri Lankan factory produces hosiery and hosts approximately 150 cells, each of which produces a single type of product on any given day. Each bra requires 18 to 42 sewing operations, depending on the complexity of the product. The number of workers is typically between nine and 15 in each cell. The Calzedonia headquarters provide the ordered sequence of operations to sew a product, as well as, for each operation m, the estimated processing time $\bar{t}_m$ taken from the General Sewing Data (GSD, 2017). The initial workload allocation reflects the order of the sewing operations given by headquarters, so that the cell can be considered as a production line for algorithmic purposes.

Bra production is labour-intensive and requires skilled sewing of many components, which are typically small (Hardaker and Fozzard, 1997). While Sri Lankan workers achieve high levels of productivity and sewing standards, it is common to encounter absenteeism (Arai, 2006; Kelegama, 2009; Wickramasinghe and Wickramasinghe, 2011). Most of the sewing

workforce is composed of young females who tend to end their career once they get married. They frequently miss work days due to family events (e.g., weddings and funerals) or personal circumstances, without informing the company. This leads to some sewing operations having no worker to complete them. It has been estimated that 7% of the workers are absent from work, without notice, on any given day. On average, this means that each production line is missing one worker every morning, and leads to a drop of 40 to 50% in productivity in the first hours of the day. The company estimated that, before our collaboration, the average time necessary to complete the factory's daily workload reallocation was one hour and 40 minutes (18% of the working day). The previous workload allocation was performed manually by line managers, each being responsible for eight production lines.

Finally, a preliminary analysis conducted by the company highlighted that the employee turnover rate is approximately 50% per annum, leading to thousands of new employees needing to be trained every year. This adds to the variability of sewing times; a worker new to the job requires much longer on the same task than an experienced worker. Considering the variability in jobs' process times is therefore crucial to obtaining good-quality solutions.

The WAP therefore models the problem encountered every morning by line managers in the factory. Whenever one or more workers are absent from a cell, the ordered sequence of jobs has to be reassigned among the workers, considering their skill levels. Workload allocations have to be planned quickly, so that high levels of efficiency can be reached from the start of the working day, and workers can either process the work-in-progress from the previous day, or start processing a new product if there is no work-in-progress inventory.

2. Literature review

Brucker et al. (2011) and De Bruecker et al. (2015) provide overviews of the personnel scheduling literature. The problems reviewed capture the needs of shift creation, staffing, and scheduling. There also exists a large body of literature on workload allocation and how the workers' skills should be taken into consideration.

In our industrial setting, each cell in the factory is an assembly system, given that a batch of products and workers move from one operation to the next, until the batch of products is completely assembled. More precisely, a single-model assembly line problem is studied, because a single product is assembled. The surveys by Scholl and Becker (2006), Becker and Scholl (2006), Boysen et al. (2007), Boysen et al. (2008), and Battaïa and Dolgui (2013) provide detailed literature reviews and classification schemes for assembly line balancing problems. The WAP is a special case of the single-model assembly line problem, because the "precedence graph" among jobs reduces to a line.

Fırat et al. (2016) aim to define a stable assignment between technicians and jobs. Miralles et al. (2007) and Moreira et al. (2015a) study a problem in which disabled people have to be assigned to workstations and jobs in a production line. Precedence constraints among operations are present and production times differ between workers. This problem is called the assembly line worker assignment and balancing problem, and has since been widely

studied (Miralles et al., 2008; Chaves et al., 2009; Blum and Miralles, 2011; Mutlu et al., 2013; Borba and Ritt, 2014; Vilà and Pereira, 2014; Moreira et al., 2015b). Zacharia and Nearchou (2016) extend the problem definition to include a multi-criteria objective function consisting of the cycle time and the smoothness index of the workload of the line, i.e., how evenly the workload is split between workers.

A related line of research has been conducted on the *Robust Assembly Line Balancing Problem* (RALBP) and its variants, which aim to incorporate the heterogeneity of the processing times through uncertainty, usually expressed as an interval for each processing time. Although the resulting solutions are robust, the algorithms require longer computation times to deal with the additional complexity. Hazir and Dolgui (2013) study the RALBP, and provide a mathematical model as well as a decomposition-based exact solution algorithm. The authors solve instances with 29 to 70 operations, for which their algorithm requires an average of 6186.29 seconds. In recent work, Pereira (2018) studies the Robust (Minmax Regret) Assembly Line Worker Assignment and Balancing Problem that involves interval type uncertainty. The author provides a formulation based on that of Borba and Ritt (2014), and presents both exact and heuristic algorithms. Through computational experiments, the author demonstrates the performance of the algorithms for instances with over 50 tasks and 7 workers. Finally, Pereira and Álvarez Miranda (2018) provide a branch-and-bound algorithm to solve the RALBP, and demonstrate that their algorithm outperforms that of Hazir and Dolgui (2013).

The workload allocation problem of Calzedonia features some characteristics of this literature, but to the best of our knowledge, no problem exactly matches the WAP. In addition, the instance sizes we aim to handle are much larger than those in the literature. Thus, the minimum number of jobs in a real instance is 630, whereas the computational reach of the formulations of Borba and Ritt (2014) are limited to instances with up to 28 tasks, the branch-and-bound algorithm of Vilà and Pereira (2014) was tested for instances with up to 75 tasks, and the computational testing of the exact algorithms of Moreira et al. (2015b) are performed on instances with up to 100 tasks.

3. Mathematical model

In order to solve the WAP, we develop a two-index mathematical model called $2I$. In this formulation, the two-index binary variable x_{ik} assumes value 1 if and only if worker $k \in W$ performs job $i \in J$. The formulation $2I$ is as follows:

$$
\begin{aligned}
(2I)\ \text{minimise} \quad & z && && (1)\\
\text{subject to} \quad & z \geq \sum_{i \in J} t_{ik} x_{ik} && k \in W \quad && (2)\\
& \sum_{k \in W} x_{ik} = 1 && i \in J \quad && (3)
\end{aligned}
$$

$$x_{ik} + x_{jk} \leq x_{hk} + 1 \qquad h, i, j \in J : i < h < j \qquad k \in W \tag{4}$$

$$x_{ik} \in \{0, 1\} \qquad i \in J, k \in W \tag{5}$$

$$z \geq 0. \tag{6}$$

The objective function (1) minimises the maximum process time, which must be at least equal to the sum of the times of jobs assigned to any worker by Constraints (2). Constraints (3) ensure that each job is assigned to a worker, and Constraints (4) ensure contiguity among the jobs assigned to a worker (a pair of jobs cannot be assigned to a worker unless all intermediate jobs are also assigned to the same worker). Constraints (5) require the assignment variables to be binary, and Constraint (6) sets the maximum process time to be non-negative.

Instances of a realistic size cannot usually be solved by a mixed integer linear programming solver using this formulation. Indeed, the number of jobs $|J|$ is typically in the order of thousands, and Constraints (4) are of order $\mathcal{O}(|J|^3 \times |W|)$.

4. A sequential exact algorithm

Once computational testing showed model $2I$ to be unable to solve medium to large sized instances, we developed an alternate exact sequential algorithm. This algorithm is based on a relaxation of the problem in which we allow workers to perform all of the jobs in an operation, a fraction of the jobs in an operation, or none of the jobs in an operation. This reduces the search space to the operations instead of the jobs, and reduces the number of constraints by multiple orders of magnitude. The compromise necessary for this reduction is that using continuous variables for the number of jobs within an operation may lead to infeasible solutions for the WAP. We therefore refer to this formulation as the Continuous Operation Relaxation Formulation (CORF).

Figure 2 depicts a solution representation of feasible workload allocations for a worker. Each square represents an operation type in the sequence. Let C denote an operation completely performed by the worker, and F denote an operation fractionally performed by the worker. The worker can completely perform a sequence of operations (Figure 2a), completely perform a sequence of operations and fractionally perform operations before or after the sequence (Figures 2b,c,d), or only fractionally perform one or two consecutive operations (Figures 2e,f).

The model CORF we have developed to solve the relaxed version of the WAP uses three new sets of variables. Let binary variable p_{mk} be equal to 1 if and only if worker $k \in W$ completely performs operation $m \in S$. The binary variable $q_{mk} = 1$ if and only if worker k fractionally performs operation m. The variable $0 \leq r_{mk} \leq 1 - \epsilon$ is the fraction of operation $m \in S$ performed by worker $k \in W$, where $\epsilon = 10^{-6}$ in our implementation. The value of r_{mk} can be greater than zero in the following two cases.

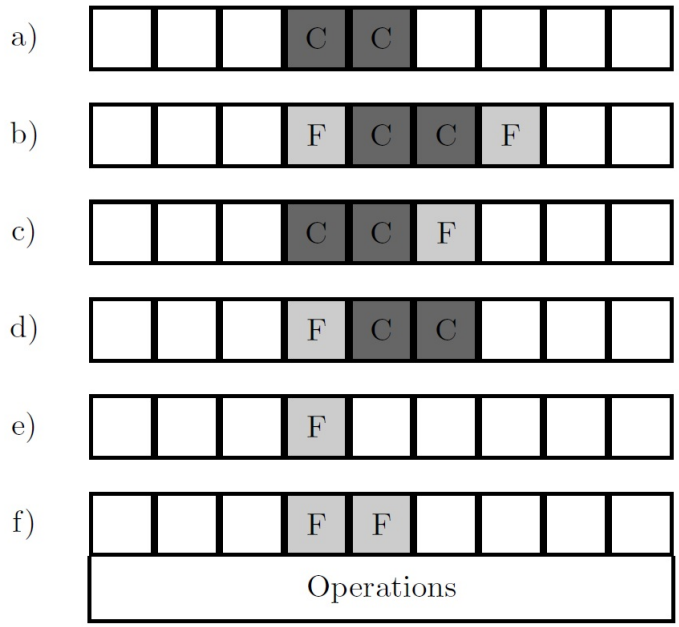

Figure 2.: Bra production in the Calzedonia Sri Lankan factory

(1) An operation m can be fractionally performed by worker k directly before operation $m+1$ which is completely performed by worker k, provided no operations $m-l$ (for $l \in \{1,\ldots,m-1\}$) are performed completely or fractionally by worker k. In other words, r_{mk} can be greater than zero if $p_{m+1,k}=1$ and $\sum_{l=1}^{m-1}(p_{lk}+q_{lk})=0$. Vice versa, an operation m can be fractionally performed by worker k directly after operation $m-1$ which is completely performed by worker k, provided no operations $m+l$ (for $l \in \{m+1,\ldots,|S|\}$) are completely or fractionally performed by worker k. In other words, r_{mk} can be greater than zero if $p_{m-1,k}=1$ and $\sum_{l=m+1}^{|S|}(p_{lk}+q_{lk})=0$. This condition allows the cases seen in Figure 2b,c,d.

(2) An operation m can be fractionally performed by worker k if no operations are completely performed by worker k, and at most one of the operations $m+1$ and $m-1$ is fractionally performed by k. In other words, r_{mk} can be greater than zero if $\sum_{m\in S} p_{mk}=0$, $\sum_{l\in S\setminus\{m-1,m,m+1\}} q_{lk}=0$, and $q_{m+1,k}+q_{m-1,k}\leq 1$. These conditions allow the cases seen in Figure 2e,f.

The formulation of CORF is

$$
\begin{aligned}
(CORF)\ \text{minimise} \quad & z && && (7)\\
\text{subject to} \quad & z \geq \sum_{m\in S}(p_{mk}+r_{mk})B\times t_{m\times B,k} && k\in W && (8)\\
& \sum_{k\in W}(p_{mk}+r_{mk})=1 && m\in S && (9)
\end{aligned}
$$

$$q_{lk} + q_{mk} + p_{lk} + p_{mk} \leq p_{ok} + 1 \qquad l, m, o \in S,\ l+1 \leq o \leq m-1;\ k \in W \qquad (10)$$

$$p_{mk} + q_{mk} \leq 1 \qquad m \in S, k \in W \qquad (11)$$

$$r_{mk} \leq q_{mk} \qquad m \in S, k \in W \qquad (12)$$

$$\sum_{m \in S} q_{mk} \leq 2 \qquad k \in W \qquad (13)$$

$$p_{mk} \in \{0, 1\} \qquad m \in S, k \in W \qquad (14)$$

$$q_{mk} \in \{0, 1\} \qquad m \in S, k \in W \qquad (15)$$

$$0 \leq r_{mk} \leq 1 - \epsilon \qquad m \in S, k \in W \qquad (16)$$

$$z \geq 0. \qquad (17)$$

The maximum process time is minimised by objective function (7) and bounded below by Constraint (8). Note the index of $t_{m \times B,k}$ instead of t_{ik} due to the fact that CORF considers operations, and t_{ik} is indexed by job. Replacing i with $m \times B$ means we are considering the time taken to complete the first job in a contiguous set of jobs of one operation. Constraints (9) require that each operation be performed. Constraints (10) impose the contiguity of operations performed for each worker and force operations fractionally performed to be either before or after a completely performed operation for worker k, if any operation has been completely performed by k. Constraints (11) ensure that a worker can perform an operation completely or fractionally, but not both. Constraints (12) link q_{mk} to the corresponding r_{mk} variable. Constraints (13) state that a worker can at most fractionally perform two operations. Note that these constraints are not necessary for the validity of the formulation (they are implied by Constraints (10), but strengthen the formulation). The domains of the variables are defined by Constraints (14) to (17).

Proposition 1. *An optimal solution value z^*_{CORF} of CORF yields a valid lower bound LB for the WAP.*

Proof. We prove Proposition 1 by showing that any feasible solution to $2I$ is also a feasible solution of CORF. A feasible solution to $2I$ is given by a matrix of binary variables x_{ik}, where $x_{ik} = 1$ if and only if worker k performs job i (as defined in Section 3). For any given feasible solution, variables x_{ik} respect Constraints (3), (4), and (5). We can then construct a feasible solution of CORF as a combination of binary variables p_{mk} and q_{mk}, and continuous variables r_{mk}. For all $m \in S$ and $k \in W$, define $f_{mk} = \sum_{i=(m-1)B+1}^{mB} x_{ik}/B$ as the fraction of operation m undertaken by worker k. If $f_{mk} = 1$ then set $p_{mk} = 1$, $q_{mk} = 0$, and $r_{mk} = 0$. If $0 < f_{mk} < 1$ then set $p_{mk} = 0$, $q_{mk} = 1$, and $r_{mk} = f_{mk}$. If $f_{mk} = 0$ then set $p_{mk} = 0$, $q_{mk} = 0$, and $r_{mk} = 0$.

This variable assignment defines a feasible solution for CORF since all constraints are satisfied. Constraints (9) are satisfied since Constraints (2) state that each job is performed

by exactly one worker. Therefore an operation can either be performed by just one worker or a combination, but the sum of assignment to the job must be one. Constraints (10) and (13) follow directly from Constraints (3), and Constraints (11) and (12) are satisfied by our transformation conditions. Since any solution of $2I$ is a solution to CORF, Proposition 1 holds. □

Note that CORF becomes a valid formulation for WAP if the values of the r_{mk} variables are restricted to integer values. This can either be imposed by cutting planes or branching constraints. However, the complexity of the formulation would increase and the problem would become intractable for realistic sized instances.

4.1. Computing an upper bound for WAP from any feasible solution of CORF

Once a solution to CORF is found, regardless of optimality, we wish to convert the solution of the relaxation into a feasible solution to WAP. A heuristic is used to convert the best objective solution returned by CORF into an upper bound for the original problem. For each operation m, the upper bound computation heuristic (CORF-H) identifies a worker k who performs a fraction of operation m (i.e., $r_{mk} > 0$). If worker k performs part or all of operation $m+1$, then worker k is marked as *lastWorker*, so their assignment for operation m will be rounded down to the closest integer number of jobs. Any other worker who performs part of operation m is then added to a vector A. The worker identified previously as the last worker is then added to the end of A. We then sequentially round the assignment of any worker up to the closest integer number of jobs, if it is not already integer. The corresponding fractional increment of workload is subtracted from the next worker in A. This process is reiterated for each worker in A. A pseudocode for this heuristic is shown in Algorithm 1. In this way we ensure that no worker can have more than one job rounded up, and the increase to the maximum process time is limited as described in Proposition 2.

Proposition 2. *Given a feasible solution of CORF with objective function value z_{CORF}, a valid upper bound UB for WAP can be obtained, for which $UB - z_{CORF} \leq \max_{i \in J, k \in W} \{t_{ik}\}$.*

Proof. We prove Proposition 2 by showing that Algorithm 1 increases the maximum process time of the solution by at most $\max_{i \in J, k \in W} \{t_{ik}\}$. As stated at the start of Section 4, each worker can be assigned at most two fractional jobs. Referring to Figure 2, the only assignments with two fractional jobs are shown in 2b and 2f. The fractional jobs are always found at the extremities of the assignment, corresponding to the first and last jobs processed by a worker. Algorithm 1 will never round up the first job assigned to a worker since the algorithm searches for the worker within an operation who has jobs in the subsequent operation. The algorithm identifies and then labels this worker as *lastWorker* in lines 10 and 11 of Algorithm 1. By rounding up at most the last job for each worker, the maximum process time of each worker cannot increase by more than $\max_{i \in J, k \in W} \{t_{ik}\}$. The change in maximum process time from CORF to CORF-H is therefore bounded by $UB - z_{CORF} \leq \max_{i \in J, k \in W} \{t_{ik}\}$. □

Algorithm 1 Upper bound computation heuristic (CORF-H)

Require: r_{mk}, p_{mk}

for $m = 1, \ldots, |S|$ **do**
  $i = 1$
  for $k = 1, \ldots, |W|$ **do**
    if $r_{mk} > 0$ **then**
      if $m = |S|$ **then**
        $A_i = k$
        $i = i + 1$
      else
        if $r_{m+1,k} > 0$ or $p_{m+1,k} = 1$ **then**
          $lastWorker = k$
        else
          $A_i = k$
          $i = i + 1$
  if $m \neq |S|$ **then**
    $A_i = lastWorker$
  for $c = 1, \ldots, i$ **do**
    if $r_{mA_c} \times B \neq \lfloor r_{mA_c} \times B \rfloor$ **then**
      $r_{mA_{c+1}} = r_{mA_{c+1}} + r_{mA_c} - (\lfloor r_{mA_c} \times B \rfloor - 1)/B$
      $r_{mA_c} = (1 + \lfloor r_{mA_c} \times B \rfloor)/B$
return r_{mk}, p_{mk}

Given an optimal solution to CORF, a stronger proposition can be proven to hold.

Proposition 3. *If z^*_{CORF} is the optimal solution value of CORF and UB^* is the upper bound obtained using algorithm CORF-H, the optimal solution value of WAP lies in the interval $[z^*_{CORF}, UB^*]$ and $UB^* - z^*_{CORF} \leq \max_{i \in J, k \in W} \{t_{ik}\}$.*

Proof. The proof follows from Propositions 1 and 2. □

4.2. A lower bound condition on workload duration

Given a valid upper bound UB for WAP, the workload of a given worker can be disregarded if it does not allow the remaining workers to complete their jobs in a time less than UB. More formally:

Proposition 4. *Given a valid upper bound UB for WAP, the workload of worker $k \in W$ that is assigned jobs from i to j; $i \leq j; i, j, \in J$, is suboptimal if*

$$\frac{\sum_{e=0}^{i-1} \min_{k \in W} \{t_{ek}\} + \sum_{e=j+1}^{|J|} \min_{k \in W} \{t_{ek}\}}{|W| - 1} > UB. \tag{18}$$

Proof. Inequality (18) considers the jobs not assigned to worker k, namely from $0, \ldots, i-1$

and $j+1, \ldots, |J|$. These jobs are assumed to all be completed at the speed of the fastest worker for that job (i.e., the fastest process time) by the remaining $|W|-1$ workers. If the sum of the process times of these jobs is equally divided across the $|W|-1$ workers, we obtain a lower bound on the workload duration of the $|W|-1$ workers. If this lower bound on the remaining workload is greater than UB time units, then the workload assigned to k is suboptimal, because the workload of at least one of the remaining workers would exceed UB. □

4.3. Three-index formulation 3I

We now introduce a three-index formulation, which we refer to as $3I$. It uses three-index binary variables y_{ijk} equal to 1 if and only if worker $k \in W$ performs jobs from i to j, $i, j \in J : i \leq j$. The formulation is then

$$
\begin{aligned}
(3I)\ \text{minimise} \quad & z && && (19)\\
\text{subject to} \quad & z \geq y_{ijk} \sum_{u \in \{i,\ldots,j\}} t_{uk} && i,j \in J : i \leq j && \\
& && k \in W && (20)\\
& \sum_{k \in W} \sum_{i \in \{1,\ldots,e\}} \sum_{j \in \{e,\ldots,|J|\}} y_{ijk} = 1 && e \in J && (21)\\
& \sum_{i \in J} \sum_{j \in \{i,\ldots,|J|\}} y_{ijk} = 1 && k \in W && (22)\\
& y_{ijk} \in \{0,1\} && k \in W && \\
& && i,j \in J : i \leq j && (23)\\
& z \geq 0. && && (24)
\end{aligned}
$$

The objective is to minimise the maximum process time (19), in conjunction with Constraints (20), which force the duration of each sequence of jobs performed by a worker to be less than or equal to z. Constraints (21) impose that each job must be assigned to a single sequence of jobs. Note that overlapping sequences are suboptimal. Constraints (22) state that each worker is assigned to one sequence of jobs. Constraints (23) and (24) define the domains of the variables.

Model $3I$ is a valid formulation for WAP, but its large number of variables makes the formulation impractical for real-sized instances. However, the number of variables can be easily reduced based on the workload duration upper bound found by CORF-H and the workload duration lower bound defined in Section 4.2. More precisely, any y_{ijk} leading to a workload longer than the upper bound obtained by CORF-H is eliminated from the formulation, as well as any y_{ijk} for which inequality (18) is satisfied. This is done by setting the upper and lower bounds of the variable to zero in our chosen solver.

4.4. Exact sequential algorithms

In summary, our 3-Component Sequential Algorithm (3CSA) consists of the following steps:

(1) Solve CORF and obtain a valid UB using CORF-H.
(2) Find suboptimal workloads according to Inequality (18).
(3) Solve $3I$, removing the variables with associated workload longer than UB or satisfying Inequality (18).

To further strengthen this algorithm, CORF can be initialised with a heuristically calculated upper bound. If this is done we refer to the algorithm as 4CSA.

5. Heuristics

We now provide the details of the heuristic implemented at the company, and we describe an Iterated Local Search metaheuristic we have developed.

5.1. Heuristic implemented by Calzedonia (IBAH)

The exact algorithms presented in Sections 3 and 4 were designed in order to assess the performance of the heuristic we implemented in Calzedonia's Sri Lankan factory. This heuristic is based on the previously used manual allocation procedure and is described below.

When performing initial workload allocation, Calzedonia headquarters calculate the size of the batch B as follows. Given the set of workers W, the set of operations S, and the estimated process time $\bar{t}_m, \forall m \in S$, given by the GSD sewing time (GSD, 2017), the batch size can be calculated as

$$B = \left\lfloor \frac{60 \times |W|}{\sum\limits_{m \in S} \bar{t}_m} \right\rfloor . \tag{25}$$

Equation (25) estimates how many products should be fully completed in an hour if all workers sew at the GSD speed. The value of B typically varies between around 35 and 120 products.

The company allocates the workload by creating a set of sequential blocks of jobs. These blocks contain the maximum number of jobs that are estimated to be completed in one hour, based on the GSD times. Each new block continues from the final job of the previous block. Next, blocks of jobs are assigned to workers by the line manager by solving a Linear Bottleneck Assignment Problem (Burkard et al., 2012) (LBAP). Hereafter we refer to the heuristic used in the factory as IBAH. The algorithm runs in a fraction of a second, therefore allows the line managers to obtain a workload allocation very quickly whenever they are made aware of absences. This allows the production to be effective from the early hours of

the morning. However, the performance of IBAH was never assessed in comparison to other methods.

5.2. Bisection Iterated Local Search (BILS)

Our computational results illustrate that the algorithm used by Calzedonia does not return satisfactory solutions, but the long running times of our exact methods are prohibitive for the industrial application. We therefore developed a metaheuristic which runs in much less time than the exact methods, and provides much better solutions than IBAH.

The metaheuristic adopts a simplified solution representation by encoding only the order of workers. A sequence of workers is converted to a full solution with assigned tasks by Algorithm 2. This bisection algorithm sequentially allocates tasks to workers. Each worker is assigned the longest possible workload below m_k units of time. The value of m_k is iteratively set to be the average of the best known upper bound UB and lower bound LB on the maximum process time. Algorithm 2 runs in pseudo-polynomial time but the procedure proves fast in practice. LB is the sum of the fastest worker's process time for every task, divided by the number of workers. UB is initialised as the sum of the slowest worker's process time for each task, and is updated by setting it to the best known maximum process time.

The metaheuristic follows the structure of Iterated Local Search presented by Lourenco et al. (2003) and sketched in Algorithm 3. The initial solution is a random sequence of workers. The local search consists of two moves: a relocation of one worker in the sequence, and a swap of two workers' positions in the sequence. Each neighbourhood comprises of $|W|^2$ moves, however we only investigate moves affecting the position of the worker with the maximum process time. This reduces the number of moves in the swap neighbourhood to $|W|$, and in the relocate neighbourhood to $2k^*(w+1)-2(k^*)^2-2$ where k^* is the position in the sequence of the worker with the maximum process time. Further details are provided in Appendix A.

For the perturbation stage of the ILS there are four moves, the first two are a random move in the local search neighbourhoods. The third is repositioning a contiguous subsequence of workers of minimum length two and maximum length $|W|-1$. The final perturbation is reversing the order of a contiguous subsequence of workers of minimum length four and maximum length $|W|$. Each perturbation move has the same probability of being selected, and each move within each perturbation neighbourhood has the same chance of being implemented. We select N_p random perturbation moves at each ILS iteration.

We tested with termination time limits of 30 seconds, one minute, and two minutes. The metaheuristic is henceforth referred to as the Bisection Iterated Local Search (BILS).

6. Computational Results

To test our algorithms we generated two sets of 100 instances each, in which all parameters were based on ten sample instances provided by Calzedonia. In both sets the number of

Algorithm 2 Bisection Algorithm

Require: $UB, LB, SeqW$
$i = 1$
while $UB - LB > 0$ **do**
  Assign a capacity of $m_k = (UB + LB)/2$ to each worker $k \in W$
  for $k = 1$ to $|W|$ **do**
    while $i \leq |J|$ **and** $m_{SeqW(k)} \geq t_{i,SeqW(k)}$ **do**
      Assign job i to worker $SeqW(k)$
      $m_{SeqW(k)} = m_{SeqW(k)} - t_{i,SeqW(k)}$
      $i = i + 1$
  if $i > |J|$ **then**
    $LB = (UB + LB)/2$
  else
    $UB = ((UB + LB)/2) - \min_{k \in W} m_k$
return UB

Algorithm 3 Iterated Local Search

$s_0 = GenerateInitialSolution$
$s^* = LocalSearch(s_0)$
while Termination condition not met **do**
  $s' = Perturb(s^*)$
  $s^{*\prime} = LocalSearch(s')$
  $s^* = AcceptanceCriterion(s^*, s^{*\prime})$

operations is $|S| \in \{18, \ldots, 42\}$ and there are four instances for each value of $|S|$. Each operation m is assigned an estimated process time $\bar{t}_m$ in minutes, by sampling from the normal distribution $N(0.4, 0.2)$, where any values smaller than 0.1 are rounded up to 0.1. The number of workers is generated by sampling from the discrete uniform distribution $U_d[9, \ldots, 15]$. Given these values, the batch size for each instance is calculated using Equation (25).

The sets of instances differ in their procedure for generating the t_{ik} values. In our first set, which we name *Average Performance*, all workers are considered to perform their jobs in a time reasonably close to the estimated value $\bar{t}_m$. Our second instance set involves a greater variation in worker skill; we name this set *Mixed Performance*. The t_{ik} values for Mixed Performance are generated by first assigning a skill level to each worker by sampling from the discrete uniform distribution $U_d[0, 2]$. Table 1 demonstrates the rules for t_{ik} generation for each instance set, and each skill level within MP. The instances are available for download at `https://people.bath.ac.uk/mb2182/instances/`.

All methods were coded in C++, and CPLEX 12.7.1 was used for solving the MILPs. Tests were run using Balena High Performance Computing (HPC) Service at the University of Bath. Full technical specifications can be found at `https://www.bath.ac.uk/bucs/services/hpc/facilities/`. In what follows, we present aggregate results for the sake of readability. We provide the detailed results in Appendix B. We also note that while the algorithms for ALWABP may be used to solve the WAP, it is unlikely that they perform better, due to the differing features of the WAP that we exploit in our algorithms. A full

Instance set	Skill level	t_{ik}
Average Performance	N/A	$\max\{\bar{t}_{\lfloor i/B \rfloor} + N(0, 0.5), 0.1\}$
Mixed Performance	0	$\max\{\bar{t}_{\lfloor i/B \rfloor} + N(0, 0.5), 0.1\}$
	1	$\max\{\bar{t}_{\lfloor i/B \rfloor} + N(-0.1, 0.3), 0.1\}$
	2	$\max\{\bar{t}_{\lfloor i/B \rfloor} + N(0.1, 0.3), 0.1\}$

Table 1.: Calculation of t_{ik} for each instance type.

computational comparison of the algorithms for the ALWABP and the WAP is beyond the scope of this paper.

6.1. Size of formulations

To give an indication of the difficulty of each method, we first present the number of variables and constraints in each of the four mathematical models. These figures are stated in Table 2 and report the average number of variables or constraints across all 200 instances.

Model	Number of variables	Number of constraints
$2I$	22,481	13,704,843,540
$3I$	21,266,535	21,268,348
$3I$(3CSA)	2,066,537	2,068,351
$3I$(4CSA)	1,812,106	1,813,920

Table 2.: Number of variables and constraints in each model.

We observe that $2I$ has a number of variables at least two orders of magnitude smaller than any other method. The number of constraints, on the other hand. is the largest of any method due to Constraints (3). We also see that using CORF to calculate upper and lower bounds to eliminate variables from $3I$ reduces the number of variables and constraints by an order of magnitude (3CSA). The reduction is more pronounced for 4CSA, which uses the lower bound of CORF and the upper bound of IBAH.

6.2. Performance of the exact algorithms

Tables 3 and 4 summarise the following results:

- **Gap(%):** The average percentage gap between the best lower bound LB and best upper bound UB provided by the solver over all instances for which the solver could provide a lower bound and an upper bound, but were not solved to optimality. Gap(%) is calculated as $\frac{UB-LB}{LB} \times 100$
- **Solved:** The number of instances out of 100 for which the method was able to conclude

to optimality within the given computing time.

- **Time(s):** The average time, in seconds, taken to find an optimal solution, calculated using only the instances solved optimally.
- **UB−LB:** The average time unit difference between the lower bound provided by CORF, and the upper bound provided by CORF-H.

Initial testing indicated that both $2I$ and $3I$ were unable to solve instances with the batch size B defined by Equation (25). We therefore ran both with batch size $\hat{B} \in \{5, 10\}$ to analyse their performance, we additionally tested $3I$ with $\hat{B} = 20$ as it had exhibited good results for $\hat{B} = 10$. Both formulations were allowed a maximum CPU time of one hour.

	Instance set	B	Gap(%)	Solved	Time(s)
$2I$	Average Performance	5	166.74	1	609.43
		10	296.14	0	N/A
	Mixed Performance	5	125.31	0	N/A
		10	230.38	0	N/A
$3I$	Average Performance	5	N/A	100	163.33
		10	2003.99	53	1401.28
		20	2857.84	0	N/A
	Mixed Performance	5	N/A	100	161.78
		10	2140.84	61	1413.05
		20	2961.99	1	3211.70

Table 3.: Results of testing $2I$ and $3I$

From Table 3 we see that $3I$ is able to solve more instances than $2I$, but as the batch size increases it becomes unable to converge to optimal solutions within the allowed time. However, the percentage gaps between bounds for unsolved instances are lower for $2I$. Algorithms 3CSA and 4CSA are able to complete instances with batch size B defined by Equation (25). For these algorithms we allow one hour of CPU time per formulation, so a full run of 3CSA or 4CSA is allowed two hours of CPU time.

		CORF			CORF–H	$3I$ with reduced variables		
	Instance set	Gap(%)	Solved	Time(s)	UB−LB	Gap(%)	Solved	Time(s)
3CSA	Average Performance	157.19	20	574.67	18.65	146.51	20	1644.14
	Mixed Performance	112.10	19	850.92	16.48	111.04	23	1432.50
4CSA	Average Performance	76.72	38	524.13	9.10	107.24	34	1485.89
	Mixed Performance	69.14	33	445.27	9.92	86.56	31	1549.55

Table 4.: Results of testing 3CSA and 4CSA

Table 4 summarises the results of our computational tests on 3CSA, 4CSA. Clearly, initialising CORF with the solution of our heuristic as an upper bound yields a benefit in the solving of CORF, and this improvement leads to a greater number optimal solutions produced by the reduced $3I$, since the tighter bounds allow more variables to be eliminated.

From both Table 3 and Table 4 we see that the performance of our exact algorithms changes little depending on the instance sets Average Performance and MP.

6.3. Performance of the metaheuristic

We have run BILS 10 times on each of the 200 instances. We have used three different time limits for BILS, 30 seconds, one minute, and two minutes, to better observe the trade-off between the CPU time requirement and the performance. Tables 5 and 6 summarise the results of our testing of BILS. Table 5 displays the deviation with respect to the best lower bound for instances of 18 to 24 operations, for which the lower bound is tight enough to conclude optimality for most instances. The column heading "Avg." refers to the aggregate average deviation from the best known solution for each instance set, whereas "Best" refers to the average of the best deviation achieved by any run of BILS for each instance. Table 6 presents the deviations with respect to the best known solution, for all instances. We emphasize that of BILS outperforms all the other algorithms we have presented.

		BILS(30 seconds)		BILS(1 minute)		BILS(2 minutes)	
Instance set	IBAH	Avg	Best	Avg	Best	Avg	Best
Average Performance	483.12	2.28	0.58	1.85	0.65	1.69	0.57
Mixed Performance	345.96	2.31	1.39	1.99	1.36	1.78	1.36

Table 5.: Average percentage deviation of the heuristic results from the best lower bound, for instances with up to 24 operations.

		BILS(30 seconds)		BILS(1 minute)		BILS(2 minutes)	
Instance set	IBAH	Avg	Best	Avg	Best	Avg	Best
Average Performance	372.48	1.76	0.05	1.35	0.04	0.90	0.02
Mixed Performance	279.02	1.05	0.05	0.84	0.07	0.48	0.00

Table 6.: Average percentage deviation of heuristic results from the best-known solution value.

6.4. Overall performance

Finally, we assess the reduction in solution value from IBAH to 4CSA or BILS with a two minute termination condition. These results are presented in Table 7. For the 4CSA solution value we took the average best known feasible solution at the conclusion of each stage of the algorithm for each instance set, and for BILS we present the best solution found from the ten runs. This gives the values stated under 'Average solution value (minutes)'. The results obtained from the full run of 4CSA and BILS are better than those found by IBAH, by approximately 70% in both of the instance sets. This enabled us to show Calzedonia that

	Average solution value				Average reduction from IBAH to:	
Instance set	IBAH	CORF-H	4CSA	BILS(2 minutes)	4CSA	BILS(2 minutes)
Average Performance	121.53	29.96	29.75	27.47	75.52	77.40
Mixed Performance	110.24	33.64	33.53	30.19	69.59	72.61

Table 7.: Comparisons of feasible solution values at each stage of 4CSA, where the solution values are in minutes.

their workload assignment strategy yields far from optimal solutions.

7. Conclusions

We have introduced, modeled and solved a new workload allocation problem which arises in practice in the apparel industry. We provided details of the manual allocation methods used by Calzedonia before this study, the heuristic algorithm IBAH. We proposed a mathematical model which can be used to optimally solve the workload allocation, but the model is impractical for real instances. We therefore presented a sequential exact algorithm, based on a relaxation of the original problem, and a metaheuristic algorithm.

Computational results evaluate the efficacy of each method, along with its respective positive and negative qualities. The heuristic IBAH offers production managers the capacity to react and solve short-term issues caused by absenteeism (or other sources of uncertainty in production scheduling) by reallocating rapidly and efficiently the workload, despite not being optimal, while the exact methods can solve small to medium sized instances to optimality within two hours. The metaheuristic algorithm we have developed provides high-quality solutions within very small computing times. It is vastly superior to IBAH and can be applied in an industrial setting.

Our study provides an original contribution to the literature by analyzing a relevant problem encountered in Calzedonia's operations, which has received less attention in workload allocation/balancing studies. In addition, our study offers exact solution methods with high efficiency, and important managerial implications.

We believe that the proposed tool has the opportunity to be adapted to the different Calzedonia production contexts in order to solve country-specific issues. This adaptation represents an interesting future development for our study. Another opportunity for future research is represented by the adaptation of the proposed algorithm not only to facilitate the reallocation in production contexts characterised by an uneven balance of skills among the workers, but also as a tool to support skills improvement and additional focused training. In this context, our exact algorithms could be of practical use, given that computing times are not as stringent in the production planning phases. The algorithm development process highlighted the important role played by workers' skills in the workload reallocation. We believe that future research would provide beneficial improvements in terms of

social sustainability in the textile industry, especially considering production contexts such as developing countries.

Acknowledgements

This work was partially supported by the Canadian Natural Sciences and Engineering Research Council under grant 2015-06189. This support is gratefully acknowledged. This research made use of the Balena High Performance Computing (HPC) Service at the University of Bath. Thanks are due to the referees for their valuable comments.

Appendix A. Proof of neighbourhood size

We aim to count the solutions in the relocate neighbourhood which involve a change in the position of the worker with the maximum process time located at position k^*. Here, $|W|$ is the number of workers.Consider a relocate move that places the worker from position $\hat{k}$ to position $\bar{k}$, where $\hat{k} \neq \bar{k}$. The worker in position k^* is affected by the move in three cases:

(1) $k^* = \hat{k}$, then $\bar{k}$ can take any value different than k^*. The number of moves to be explored is $|W| - 1$.
(2) $\hat{k} < k^* \leq \bar{k}$, then $\hat{k}$ can take $k^* - 1$ values and $\bar{k}$ can take $|W| + 1 - k^*$ values. The number of moves to be explored is $(k^* - 1)(|W| + 1 - k^*)$.
(3) $\bar{k} \leq k^* < \hat{k}$, then $\hat{k}$ can take $|W| - k^*$ values and $\bar{k}$ can take k^* values. The number of moves to be explored is $k^*(|W| - k^*)$.

So the total number of moves is $|W| - 1 + (k^* - 1)(|W| + 1 - k^*) + k^*(|W| - k^*)$ which simplifies to $2k^*(|W| + 1) - 2(k^*)^2 - 2$ as stated in the main text.

Appendix B. Detailed computational results

Instance	$\|J\|$	$\|W\|$	LB	Best BILS(2m)	Average BILS(2m)
1	18	15	13.39	13.45	13.83
2	19	9	**31.52**	**31.52**	**31.52**
3	20	15	19.63	19.72	20.98
4	21	15	**21.15**	**21.15**	21.31
5	22	12	**21.99**	**21.99**	22.14
6	23	10	**29.09**	**29.09**	29.14
7	24	12	**21.50**	**21.50**	21.50
8	25	14	26.76	27.06	27.86
9	26	11	**25.59**	**25.59**	25.78
10	27	13	20.67	24.40	24.40
11	28	11	18.02	26.96	26.96
12	29	14	**25.23**	**25.23**	25.44
13	30	10	24.10	31.08	31.08
14	31	14	**18.42**	**18.42**	18.87
15	32	11	23.45	31.87	31.87
16	33	15	19.35	23.41	23.42
17	34	9	20.74	32.82	32.82
18	35	13	20.73	30.77	31.35
19	36	15	16.77	23.01	23.08
20	37	13	17.43	24.73	24.75
21	38	14	20.77	28.95	29.05
22	39	15	17.92	23.98	23.98
23	40	10	18.29	42.32	42.32
24	41	10	17.42	36.59	36.59
25	42	15	24.17	32.82	33.12
26	18	10	**29.05**	**29.05**	**29.05**
27	19	12	**20.13**	**20.13**	20.50
28	20	11	**20.39**	**20.39**	**20.39**
29	21	9	**28.89**	**28.89**	**28.89**
30	22	10	**34.19**	**34.19**	**34.19**
31	23	11	**30.47**	**30.47**	**30.47**
32	24	14	20.37	20.37	20.88
33	25	11	**23.26**	**23.26**	**23.26**
34	26	10	31.02	34.37	34.37
35	27	14	21.18	21.24	21.29
36	28	14	**22.82**	**22.82**	**22.82**
37	29	11	24.44	33.37	33.37
38	30	10	21.74	29.95	29.95
39	31	14	15.71	18.80	18.91
40	32	15	23.71	26.59	26.95
41	33	14	28.24	28.34	28.50
42	34	10	**36.51**	**36.51**	**36.51**
43	35	9	17.99	37.51	37.51
44	36	9	19.36	39.42	39.42
45	37	10	17.74	33.91	33.91
46	38	15	18.68	29.21	29.94
47	39	15	17.61	22.34	22.58
48	40	10	17.68	33.69	33.69
49	41	9	18.00	37.45	37.45
50	42	15	12.38	25.94	27.15

Table B1.: Detailed results for instances 1–50 in the Average Performance set

Instance	$\|J\|$	$\|W\|$	LB	Best BILS(2m)	Average BILS(2m)
51	18	10	**30.49**	**30.49**	**30.49**
52	19	13	15.04	15.05	15.06
53	20	12	**23.61**	**23.61**	**23.61**
54	21	13	22.86	23.01	23.40
55	22	15	**19.43**	**19.43**	**19.43**
56	23	14	19.59	19.70	19.71
57	24	11	**27.16**	**27.16**	**27.16**
58	25	9	29.38	32.75	32.75
59	26	14	**20.89**	**20.89**	20.90
60	27	13	**26.18**	**26.18**	**26.18**
61	28	11	**28.14**	**28.14**	**28.14**
62	29	11	**26.19**	**26.19**	**26.19**
63	30	12	21.73	31.23	31.29
64	31	15	17.08	23.37	24.70
65	32	14	**22.77**	**22.77**	23.13
66	33	14	15.73	22.90	22.90
67	34	11	25.51	33.73	34.01
68	35	12	16.38	29.71	29.82
69	36	10	21.54	41.44	41.44
70	37	10	20.11	38.55	38.55
71	38	12	22.26	32.40	33.70
72	39	11	15.50	31.37	31.37
73	40	9	18.81	40.30	40.30
74	41	14	17.63	31.04	31.25
75	42	15	12.48	28.15	28.78
76	18	11	**21.05**	**21.05**	21.42
77	19	10	**17.83**	**17.83**	**17.83**
78	20	11	**23.35**	**23.35**	23.68
79	21	11	**25.26**	**25.26**	**25.26**
80	22	15	16.67	16.74	17.11
81	23	13	**15.30**	**15.30**	15.32
82	24	14	**18.57**	**18.57**	19.49
83	25	15	20.69	20.80	21.91
84	26	15	17.66	17.70	17.91
85	27	12	**27.32**	**27.32**	**27.32**
86	28	13	15.85	23.60	23.73
87	29	9	31.29	31.29	31.29
88	30	13	23.08	25.75	26.11
89	31	13	20.32	22.29	22.31
90	32	14	16.81	21.30	21.59
91	33	15	14.28	19.50	20.40
92	34	9	21.92	41.81	41.81
93	35	13	19.27	27.80	28.26
94	36	11	15.93	32.38	32.38
95	37	12	19.15	32.64	32.81
96	38	14	19.37	26.28	26.41
97	39	14	22.09	30.39	30.58
98	40	15	14.53	27.43	28.34
99	41	14	14.47	27.10	27.50
100	42	10	19.89	46.92	46.92

Table B2.: Detailed results for instances 51–100 in the Average Performance set

Instance	$\|J\|$	$\|W\|$	LB	Best BILS(2m)	Average BILS(2m)
1	18	11	**28.41**	**28.41**	**28.41**
2	19	13	28.37	28.47	28.54
3	20	13	23.80	23.91	23.91
4	21	12	27.21	27.27	27.30
5	22	14	**18.75**	**18.75**	18.97
6	23	9	**25.91**	**25.91**	**25.91**
7	24	14	22.31	23.38	23.47
8	25	12	26.01	29.13	29.42
9	26	9	**41.67**	**41.67**	**41.67**
10	27	12	**24.63**	**24.63**	24.90
11	28	11	31.10	32.94	32.94
12	29	9	**33.52**	**33.52**	**33.52**
13	30	12	**26.88**	**26.88**	**26.88**
14	31	10	28.77	39.01	39.01
15	32	14	21.87	24.95	24.96
16	33	13	**26.80**	**26.80**	27.29
17	34	13	21.70	27.09	27.95
18	35	14	26.54	31.68	32.20
19	36	10	25.91	33.69	33.69
20	37	13	20.52	23.42	23.77
21	38	13	26.88	32.24	32.24
22	39	9	21.28	38.93	38.93
23	40	14	19.01	30.68	30.92
24	41	12	34.63	34.76	34.76
25	42	10	16.08	32.09	32.12
26	18	10	**27.33**	**27.33**	**27.33**
27	19	10	**28.66**	**28.66**	**28.66**
28	20	11	**25.60**	**25.60**	**25.60**
29	21	13	27.63	27.94	28.03
30	22	9	**30.31**	**30.31**	**30.31**
31	23	12	31.62	31.77	31.77
32	24	14	24.98	28.46	28.63
33	25	13	**24.19**	**24.19**	24.21
34	26	11	**23.30**	**23.30**	**23.30**
35	27	11	24.95	33.80	33.80
36	28	11	**29.16**	**29.16**	**29.16**
37	29	14	22.95	23.05	23.47
38	30	13	22.86	35.93	36.28
39	31	10	29.31	36.32	36.32
40	32	14	22.24	29.39	29.65
41	33	10	**32.31**	**32.31**	**32.31**
42	34	15	**30.90**	**30.90**	31.00
43	35	11	22.77	32.19	32.19
44	36	14	21.89	32.58	33.12
45	37	11	23.37	37.88	37.88
46	38	12	18.90	31.35	31.35
47	39	9	22.49	42.94	42.94
48	40	14	18.06	26.98	27.31
49	41	15	16.65	32.34	32.93
50	42	10	17.69	37.65	37.80

Table B3.: Detailed results for instances 1–50 in the Mixed Performance set

Instance	$\|J\|$	$\|W\|$	LB	Best BILS(2m)	Average BILS(2m)
51	18	11	**26.27**	**26.27**	**26.27**
52	19	11	**21.11**	**21.11**	21.15
53	20	14	23.74	23.88	24.06
54	21	14	**23.10**	**23.10**	23.27
55	22	15	21.71	21.84	21.87
56	23	13	32.03	32.17	32.61
57	24	14	**22.64**	**22.64**	**22.64**
58	25	15	34.02	34.22	35.12
59	26	11	**28.63**	**28.63**	**28.63**
60	27	11	35.35	36.68	36.68
61	28	11	**24.56**	**24.56**	**24.56**
62	29	14	18.73	22.02	22.26
63	30	11	23.88	35.40	35.40
64	31	14	22.95	26.30	26.33
65	32	12	24.25	35.54	35.89
66	33	9	25.24	43.94	43.94
67	34	15	**25.88**	**25.88**	25.98
68	35	15	17.86	25.98	26.45
69	36	13	18.84	29.88	30.07
70	37	9	**36.35**	**36.35**	**36.35**
71	38	12	16.12	31.09	31.09
72	39	9	21.70	40.40	40.40
73	40	10	17.62	38.72	38.72
74	41	13	23.47	38.34	38.85
75	42	14	24.32	36.56	36.64
76	18	12	22.76	22.93	23.09
77	19	15	22.61	22.78	22.78
78	20	13	**22.95**	**22.95**	23.17
79	21	13	22.93	23.00	23.15
80	22	10	**31.36**	**31.36**	**31.36**
81	23	13	**32.60**	**32.60**	32.80
82	24	13	**23.45**	**23.45**	23.58
83	25	14	**22.80**	**22.80**	**22.80**
84	26	12	**29.90**	**29.90**	30.30
85	27	11	**22.40**	**22.40**	**22.40**
86	28	12	25.43	29.10	29.10
87	29	14	21.24	27.69	28.40
88	30	13	**25.16**	**25.16**	**25.16**
89	31	11	**26.52**	**26.52**	**26.52**
90	32	14	21.13	26.45	26.62
91	33	15	16.16	22.22	22.56
92	34	12	18.74	33.18	33.66
93	35	10	19.33	38.04	38.04
94	36	12	21.76	28.45	28.50
95	37	12	27.04	39.53	39.53
96	38	12	14.57	35.97	35.97
97	39	9	22.34	41.99	41.99
98	40	13	21.86	33.95	33.95
99	41	14	15.74	27.86	28.13
100	42	10	21.90	41.22	41.22

Table B4.: Detailed results for instances 51–100 in the Mixed Performance set

References

Arai, E. (2006). Readymade garment workers in Sri Lanka: Strategy to survive in competition. In Murayama, M., editor, *Employment in readymade garment industry in post-MFA era: The cases of India, Bangladesh and Sri Lanka*, chapter 2, pages 31–52. JRP Series no. 140. Chiba: Institute of Developing Economies.

Battaïa, O. and Dolgui, A. (2013). A taxonomy of line balancing problems and their solution approaches. *International Journal of Production Economics*, 142:259–277.

Becker, C. and Scholl, A. (2006). A survey on problems and methods in generalized assembly line balancing. *European Journal of Operational Research*, 168(3):694–715.

Blum, C. and Miralles, C. (2011). On solving the assembly line worker assignment and balancing problem via beam search. *Computers & Operations Research*, 38(1):328–339.

Borba, L. and Ritt, M. (2014). A heuristic and a branch-and-bound algorithm for the assembly line worker assignment and balancing problem. *Computers & Operations Research*, 45:87–96.

Boysen, N., Fliedner, M., and Scholl, A. (2007). A classification of assembly line balancing problems. *European Journal of Operational Research*, 183(2):674–693.

Boysen, N., Fliedner, M., and Scholl, A. (2008). Assembly line balancing: Which model to use when? *International Journal of Production Economics*, 111(2):509–528.

Brucker, P., Qu, R., and Burke, E. (2011). Personnel scheduling: Models and complexity. *European Journal of Operational Research*, 210(3):467–473.

Burkard, R., Dell'Amico, M., and Martello, S. (2012). *Assignment problems*. Society for Industrial and Applied Mathematics, Philadelphia.

Chaves, A. A., Lorena, L. A. N., and Miralles, C. (2009). Hybrid metaheuristic for the assembly line worker assignment and balancing problem. *Hybrid Metaheuristics*, 5818:1–14.

De Bruecker, P., Van den Bergh, J., Belien, J., and Demeulemeester, E. (2015). Workforce planning incorporating skills: State of the art. *European Journal of Operational Research*, 243(1):1–16.

Fırat, M., Briskorn, D., and Laugier, A. (2016). A branch-and-price algorithm for stable workforce assignments with hierarchical skills. *European Journal of Operational Research*, 251(2):676–685.

GSD (2017). General sewing data.

Hardaker, C. and Fozzard, G. (1997). The bra design process: A study of professional practice. *International Journal of Clothing Science and Technology*, 9(4):311–325.

Hazir, H. and Dolgui, A. (2013). Assembly line balancing under uncertainty: Robust optimization models and exact solution method. *Computers & Industrial Engineering*, 65(2):261–267.

Kelegama, S. (2009). Ready-made garment exports from Sri Lanka. *Journal of Contemporary Asia*, 39(4):579–596.

Li, W., Freiheit, T., and Miao, E. (2017). A lever concept integrated with simple rules for flow shop scheduling. *International Journal of Production Research*, 55(11):3110–3125.

Li, X., Ishii, H., and Chen, M. (2015). Single machine parallel scheduling problem with fuzzy due-date and fuzzy precedence relation. *International Journal of Production Research*, 53(9):2707–2717.

Lourenco, H., Martin, O., and Stutzle, T. (2003). Iterated local search. In *Handbook of Metaheuristics*, pages 320–353. Springer, Boston, MA.

Miralles, C., Garcia-Sabater, J., Andrés, C., and Cardos, M. (2007). Advantages of assembly lines in sheltered work centres for disabled. A case study. *International Journal of Production Economics*, 110(1):187–197.

Miralles, C., Garcia-Sabater, J., Andrés, C., and Cardos, M. (2008). Branch and bound procedures

for solving the assembly line worker assignment and balancing problem: Application to sheltered work centres for disabled. *Discrete Applied Mathematics*, 156(3):352–367.

Moreira, M., Cordeau, J.-F., Costa, A., and Laporte, G. (2015a). Robust assembly line balancing with heterogeneous workers. *Computers & Industrial Engineering*, 88:254–263.

Moreira, M., Miralles, C., and Costa, A. (2015b). Model and heuristics for the assembly line worker integration and balancing problem. *Computers & Operations Research*, 54:64–73.

Mutlu, O., Polat, O., and Supciller, A. (2013). An iterative genetic algorithm for the assembly line worker assignment and balancing problem of type-II. *Computers & Operations Research*, 40(1):418–426.

Pereira, J. (2018). The robust (minmax regret) assembly line worker assignment and balancing problem. *Computers & Operations Research*, 93:27–40.

Pereira, J. and Álvarez Miranda, E. (2018). An exact approach for the robust assembly line balancing problem. *Omega*, 78:85–98.

Scholl, A. and Becker, C. (2006). State-of-the-art exact and heuristic solution procedures for simple assembly line balancing. *European Journal of Operational Research*, 168(3):666–693.

Vilà, M. and Pereira, J. (2014). A branch-and-bound algorithm for assembly line worker assignment and balancing problems. *Computers & Operations Research*, 44:105–114.

Wickramasinghe, D. and Wickramasinghe, V. (2011). Perceived organisational support, job involvement and turnover intention in lean production in Sri Lanka. *The International Journal of Advanced Manufacturing Technology*, 55(5):817–830.

Zacharia, P. and Nearchou, A. (2016). A population-based algorithm for the bi-objective assembly line worker assignment and balancing problem. *Engineering Applications of Artificial Intelligence*, 49:1–9.